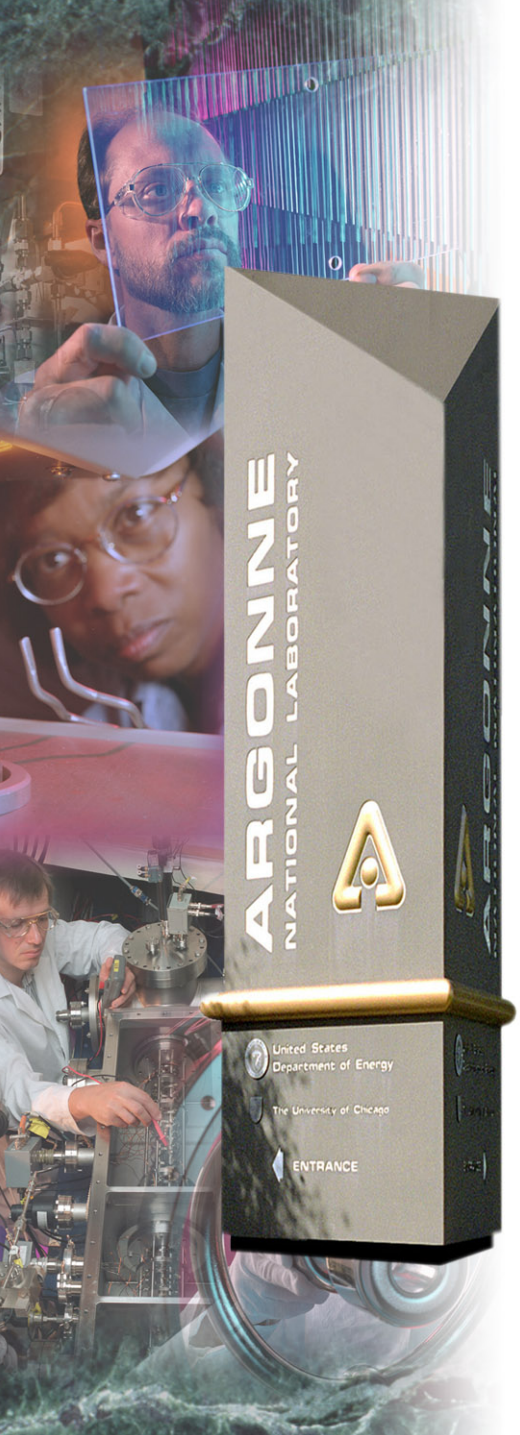



# *Tcl/Tk and the OAG Tcl/Tk Environment*

*Robert Soliday*
*Assistant Scientist / AOD*

*September 21st, 2004*

***Argonne National Laboratory***

Office of Science
U.S. Department of Energy

*A U.S. Department of Energy*
*Office of Science Laboratory*
*Operated by The University of Chicago*

# *Outline*

- **Introduction to the Tcl/Tk environment**
- **Misconceptions about Tcl/Tk**
- **Explanation of OAG Tcl/Tk**
- **Channel Access extension**
  - Loading the package
  - Explanation of commands
- **SDDS extension**
  - Loading the package
  - Explanation of commands
- **Documentation on the web**
- **How to obtain OAG Tcl/Tk for your system**

# *Introduction to the Tcl/Tk Environment*

- **Why would I want to use Tcl/Tk? What is in it for me?**
  - With Tcl (Tool Command Language) you can creating useful programs in minutes.
  - Freely available for X Windows, Windows, and Macintosh.
  - Tcl has simple constructs and looks somewhat like C.
    - *This means that it is easy read and understand the source code. This is useful when editing other people's programs.*
- **Three benefits to a script-based Tk approach to user interface programming.**
  - No compilations means fast development.
  - GUIs require just a handful of commands to define them because the Tk library provides a higher level interface than most standard C library toolkits.
  - The GUI is clearly factored out from the rest of the application.

# *Misconceptions about Tcl/Tk*

- **Perl is faster than Tcl?**
  - This was true many years ago, but Tcl caught up with the 8.0 release in 1997. This version introduced the bytecode interpreter.
- **Perl has better regular expressions?**
  - Again, this was true in the past, but the current Tcl's regular expression package has all of the advanced features Perl has.
- **Tcl doesn't support namespaces or object-oriented programming?**
  - Namespaces were added in 1997 and object-oriented programming is available in the [incr Tcl] extension.
- **Tcl can't handle binary data?**
  - Tcl has supported binary data since the 8.0 release.

# *Explanation of OAG Tcl/Tk*

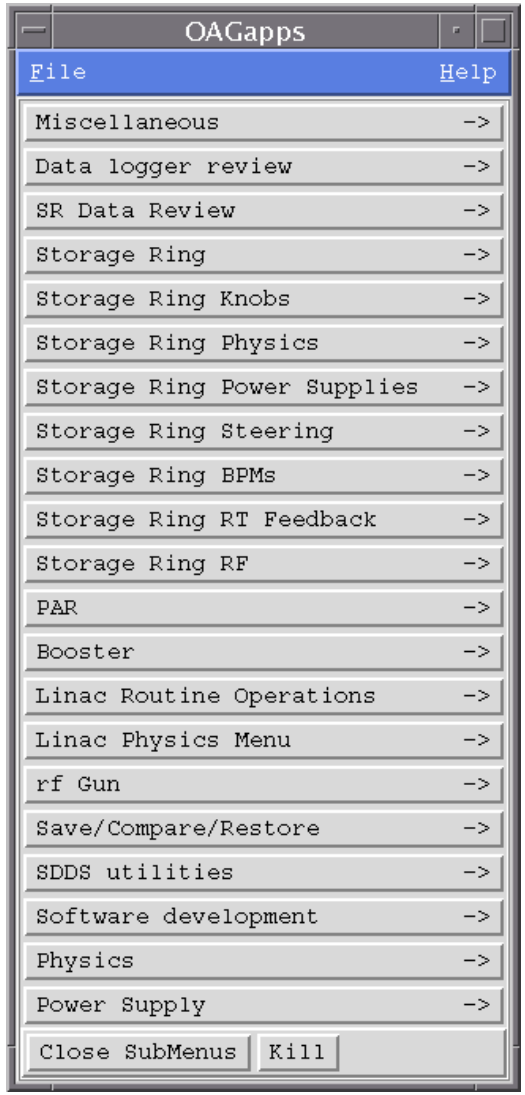
- **OAG Tcl/Tk Interpreter**
  - Extensive collection of general purpose graphical and non-graphical Tcl/Tk procedures.
    - Main windows
    - Basic widgets
    - Complex widgets
    - Utility procedures
  - Our version of Tcl/Tk includes four C extensions.
    - CA (Channel Access based originally on et_wish)
    - SDDS (Self Describing Data Sets)
    - RPN (Reverse Polish Notation)
    - OS (Miscellaneous commands)

# *Explanation of OAG Tcl/Tk*

- **10 years of development has resulted in a wide range of applications and general purpose functions.**
- **Having a GUI with a simple and powerful scripting language along with CA and SDDS extensions that works on most operating systems means we can spend less time worrying about the tools and more time working about the project.**

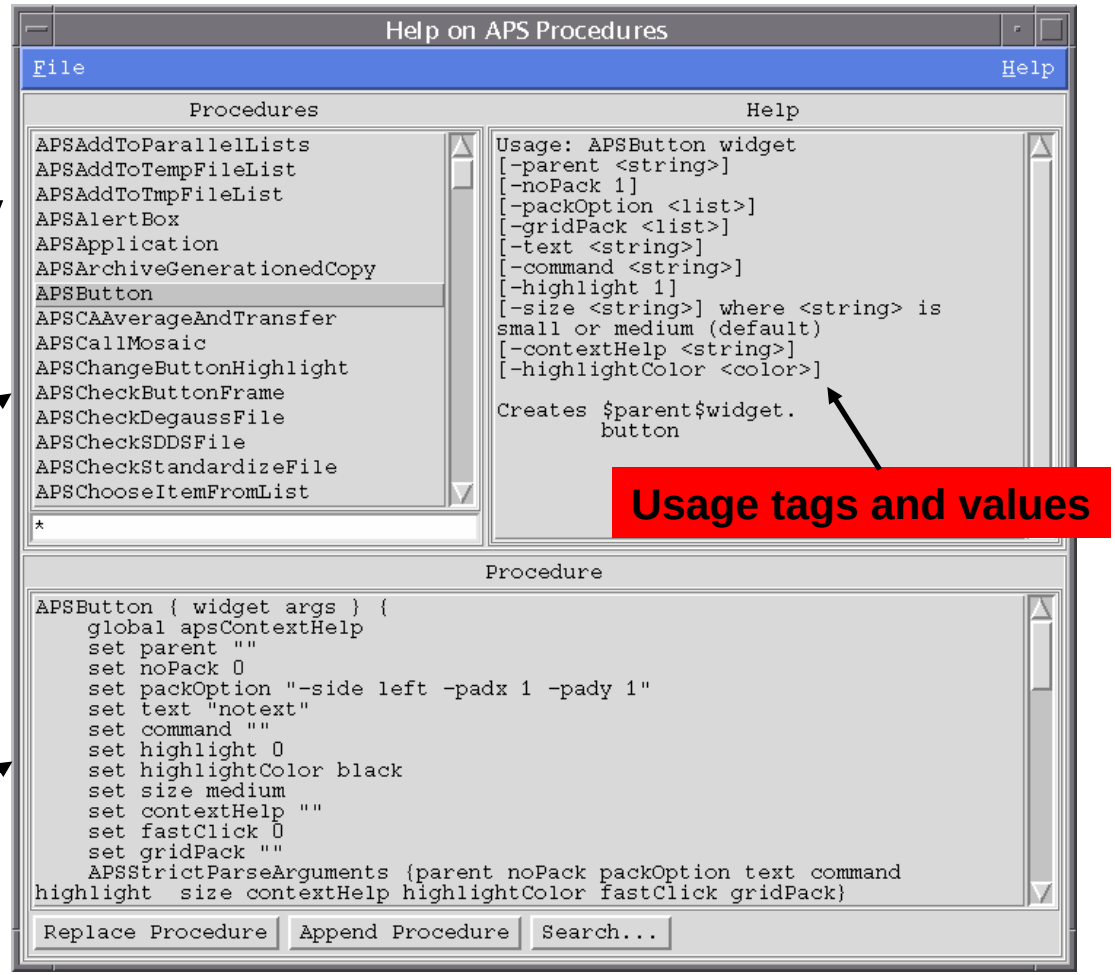
# OAG Tcl/Tk Applications

- **OAGapps**
  - Contains a large collection of OAG Tcl/Tk applications.
  - Available off the Workspace Menu under Accelerator Data.



# OAG Tcl/Tk Widgets and Procedures

- **OAG Tcl/Tk help**
  - Used as a quick reference to display the syntax of OAG Tcl/Tk widgets and procedures.



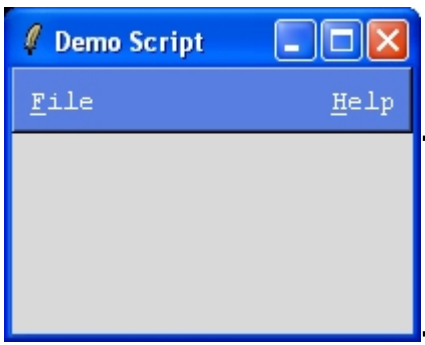
# OAG Tcl/Tk Main Window

- **Creating the main window in an OAG application.**

```
#!/bin/sh
# \
exec oagwish “$0” “$@”
set auto_path [linsert $auto_path 0 /usr/local/oag/apps/lib/$env(HOST_ARCH)]
set auto_path [linsert $auto_path 0 /usr/local/oag/lib_patch/$env(HOST_ARCH)]
APSApplication . –name “Demo Script”
```

**Used to start the Tcl/Tk interpreter**

**Used to locate OAG Tcl/Tk library functions. The TCLLIBPATH environment variable serves the same purpose on Windows computers**

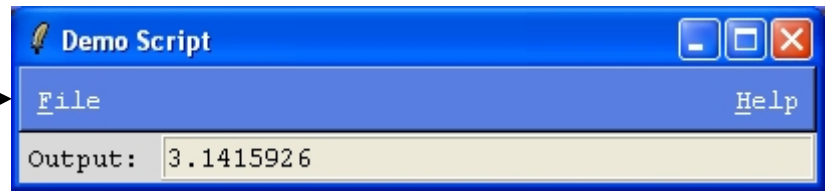


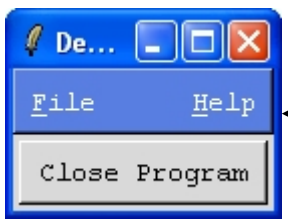

**Main frame is called .userFrame**

# OAG Tcl/Tk Basic Widgets


```
APSLabeledEntry .entry \
    -parent .userFrame \
    -label "Entry: " \
    -textVariable entryVariable \
    -width 40 \
    -contextHelp "This is a plain entry box"
```

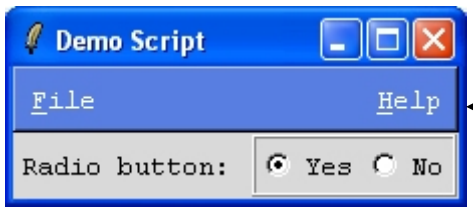
```
APSLabeledOutput .output \
    -parent .userFrame \
    -label "Output: " \
    -textVariable outputVariable \
    -width 40 \
    -contextHelp "This is a plain output box"
```




```
APSButton .button \
    -parent .userFrame \
    -text "Close Program" \
    -command {exit} \
    -contextHelp "This button closes the program."
```

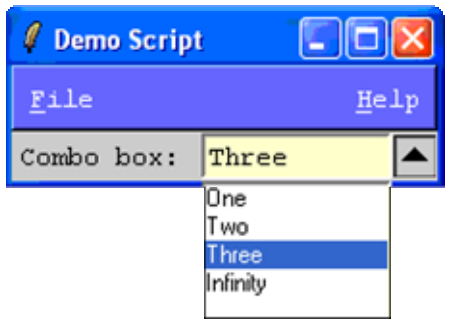
# OAG Tcl/Tk Basic Widgets




```
APSRadioButtonFrame .radiobutton \
    -parent .userFrame \
    -variable radioVariable \
    -label "Radio button: " \
    -orientation horizontal \
    -buttonList "Yes No" \
    -valueList "1 0" \
    -commandList {"Yes!" "No!"}
```

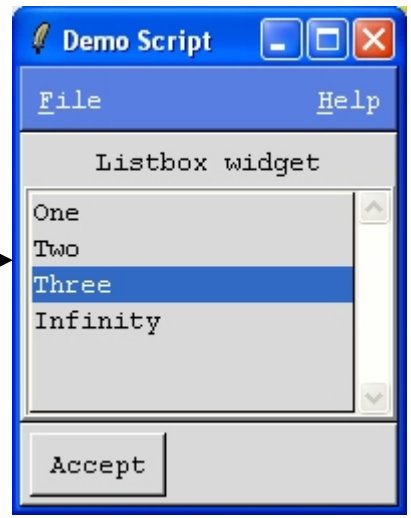
```
APSCheckButtonFrame .checkbutton \
    -parent .userFrame \
    -variableList "color(blue) color(green)" \
    -buttonList "blue green" \
    -allNone 1 \
    -label "Check buttons: " \
    -orientation horizontal
```

# OAG Tcl/Tk Basic Widgets



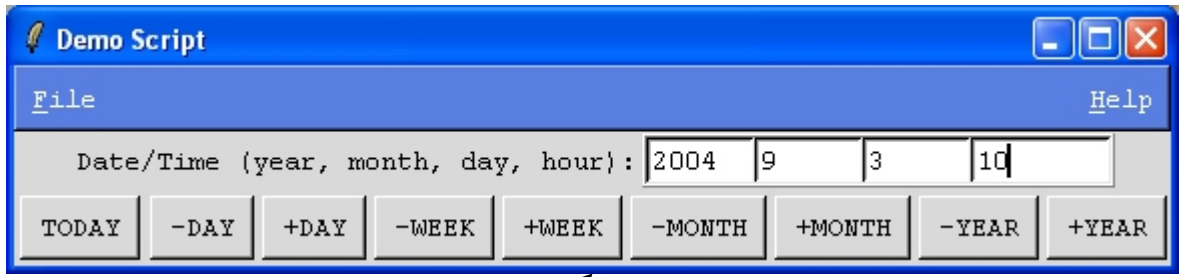

```
APSComboboxFrame .combobox \
    -parent .userFrame \
    -label "Combo box: " \
    -width 50 \
    -itemList "One Two Three Infinity" \
    -textVariable comboBoxChoice
```

```
APSScrolledListWindow .scrolledlist \
    -parent .userFrame \
    -label "Listbox widget" \
    -height 6 \
    -itemList "One Two Three Infinity" \
    -selectMode extended \
    -selectionVar scrolledListSelection \
    -closeButton 0
```



# OAG Tcl/Tk Complex Widgets

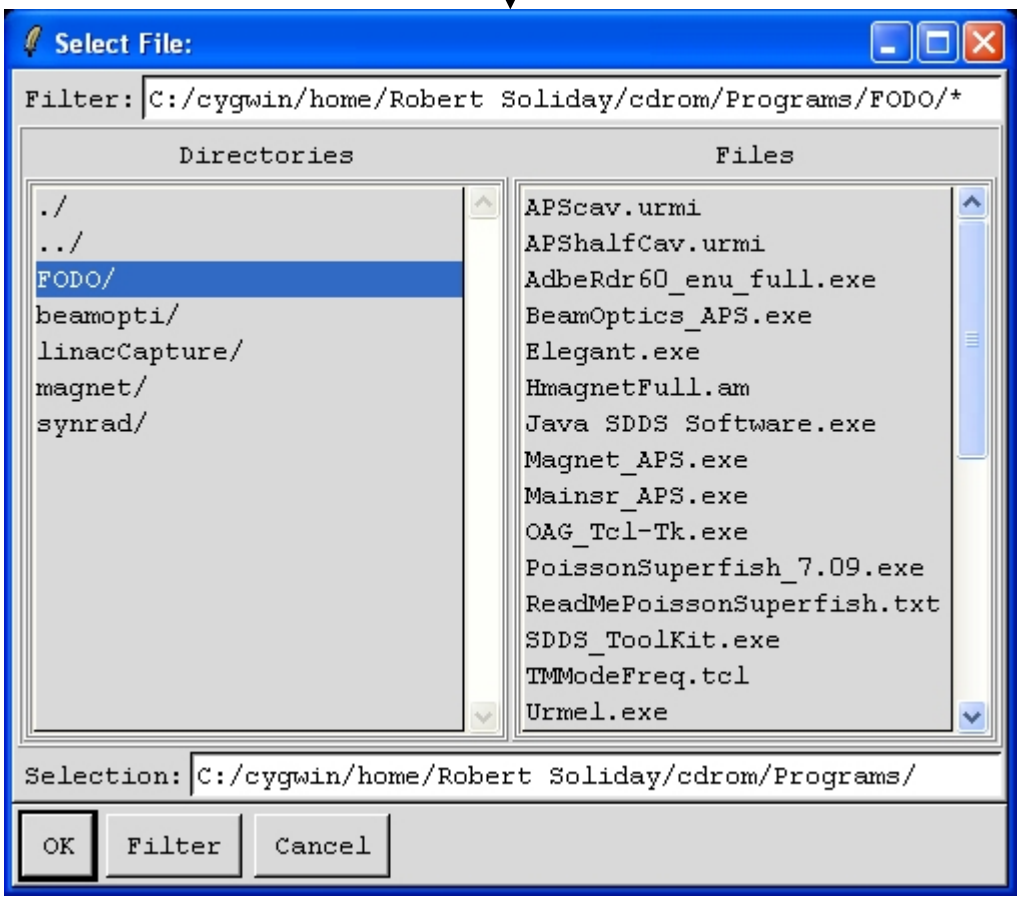

```
APSAlertBox .alert \
    -errorMessage "An error has occured" \
    -type error
```



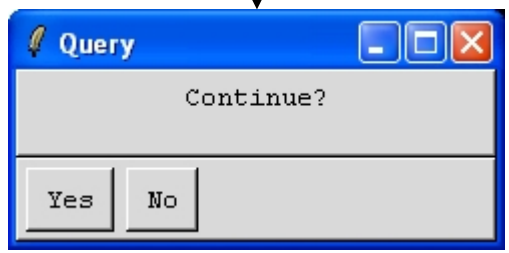

```
APSDateTimeAdjEntry .date \
    -parent .userFrame \
    -dayVariable Day \
    -monthVariable Month \
    -yearVariable Year \
    -hourVariable Hour \
    -label "Date/Time (year, month, day, hour):"
```

# OAG Tcl/Tk Complex Widgets

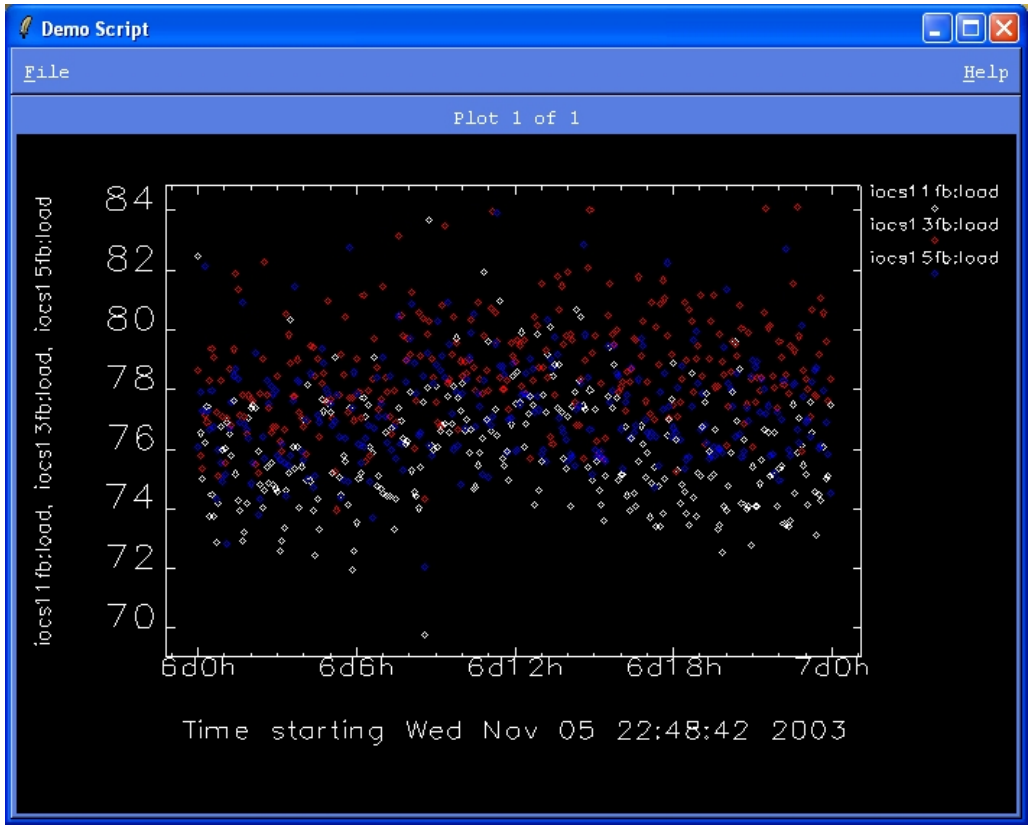
**APSFileSelectDialog .fileselect**




**APSMultipleChoice .choice **
**-question "Continue?" **
**-labelList "Yes No" **
**-returnList "1 0" **
**-name "Query“**



# OAG Tcl/Tk Complex Widgets




```
APSPlotCanvas .plot \
    -parent .userFrame \
    -sddsplotOptions “-column=Time,(iocs11fb:load,iocs13fb:load,iocs15fb:load)
        srIOC-2003-310-1106.gz -graph=symbol,vary=subtype -ticks=xtime -legend"
```

# *OAG Tcl/Tk Utility Procedures*

- **APSStandardSetup**
  - Initializes a number of global variables and Tcl/Tk options. Usually used in place of APSApplication for non-GUI applications.
- **APSStrictParseArguments**
  - Translates sequences like “-keyword value” into “set keyword value”; that is, the keyword names are variable names in the calling procedure.
- **APSWaitWithUpdate**
  - Pauses the execution but keeps the GUI responsive. Also allows for an abort variable whose change will result in the execution to continue immediately.
- **APSEnableDisableWidget**
  - Enables or disables the named widget and all of its children.

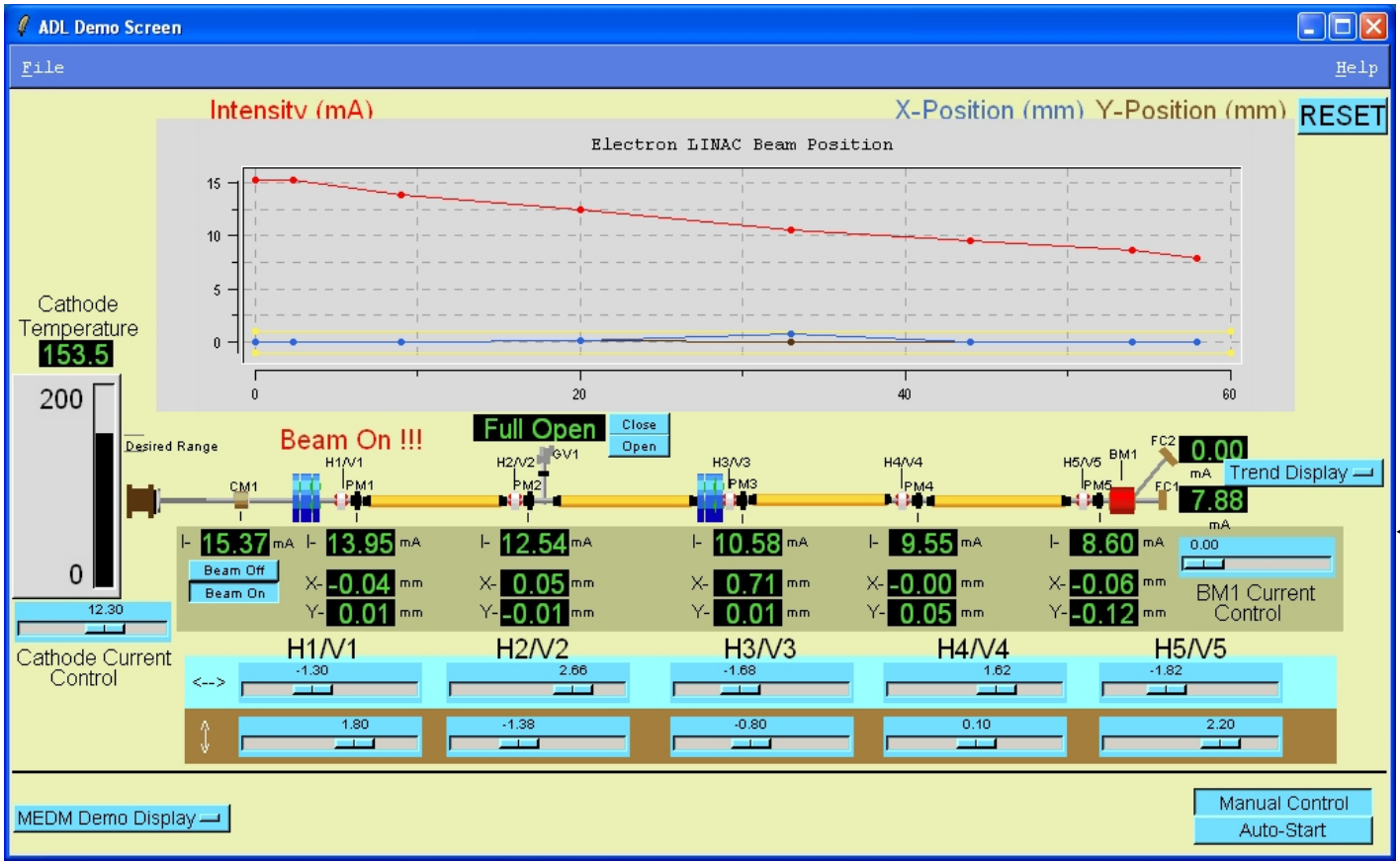
# *OAG Tcl/Tk Utility Procedures*

- **APSPrintText**
  - Options include the variable containing the text and the print command itself.
- **APSSendEMail**
  - Used to send email or page. Includes the option to not page at inconsiderate hours.
- **APScavget, APScavput**
  - Copies the functionality and command line options of cavget and cavput while using the Tcl CA Extension. The benefit being that only one CA connection is made per PV no matter how many times it is called.

# *Channel Access with OAG Tcl/Tk*

- **PV values become Tcl/Tk variables**
  - Values can be changed locally and set to an IOC, or
  - PV values in an IOC can change and be monitored by OAG Tcl/Tk
- **No external programs are required**
  - Increased speed because of the time it takes to launch an external program

# OAG Tcl/Tk Demo using CA Extension




**MEDM input file is parsed by Tcl/Tk and appropriate widgets are created and linked to the associated PVs**

**Notice Exceed is not running**

# Channel Access Extension

- Loading the CA extension into oagwish (basic method)

```
> oagwish
% package require ca
1.0
```

- Loading the CA extension into oagwish (standard method)

```
#!/bin/sh
# \
exec oagwish “$0” “$@”
set auto_path [linsert $auto_path 0 /usr/local/oag/apps/lib/$env(HOST_ARCH)]
set auto_path [linsert $auto_path 0 /usr/local/oag/lib_patch/$env(HOST_ARCH)]
APSApplication . –name “Demo Script”
```

# *Overview of CA Extension Commands*

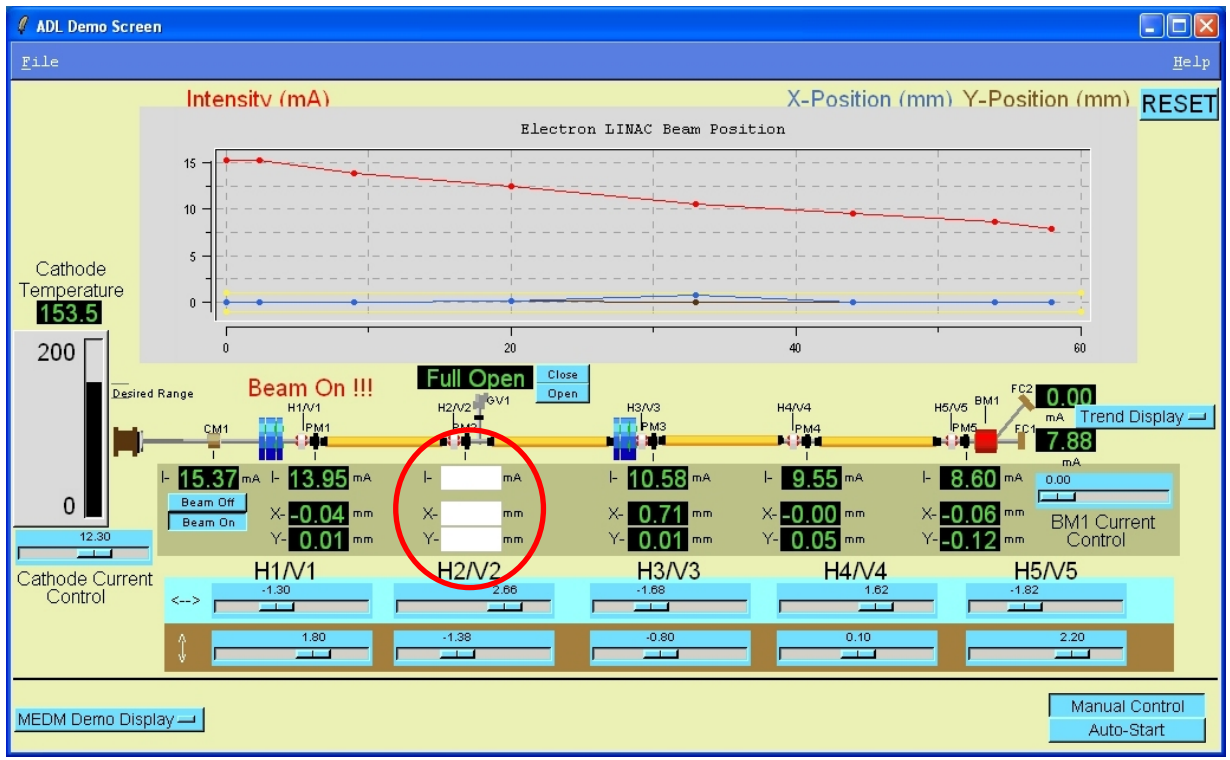
- **link, linkw, unlink**
  - Link and unlink to process variables.
- **get, getw**
  - Set related Tcl variable to the value of the PV.
- **put, putw, putq**
  - Set PV to the value of the related Tcl variable.
- **mon, umon, cmon**
  - Establish a monitor for the between the Tcl variable and the value of the PV.
- **info**
  - Obtain information about linked tcl variables.
- **stat**
  - Obtain state, status, severity and time information about a PV.
- **toggleErrorMode**
  - Toggle between standard error handling and backward compatibility with previous version of the CA extension.

# CA Link Commands

- **When should you use link instead of linkw?**
  - When you want to keep the GUI responsive.
    - *If the user is allowed to type the PV name there is a chance of a typo which would lead to long timeout periods where the GUI would appear to lockup if using linkw.*
  - When you want to find the name of a PV that is not connecting.
    - *With linkw it is not possible to isolate an invalid PV name.*
- **When should you use linkw instead of link?**
  - When you are reasonably sure the PVs will link without problems.
    - *Normally PVs will connect in under a second and GUI responsiveness does not come into play.*
  - When ease of use is desirable.
    - *It blocks execution until all the PVs have connected.*

# CA Link Commands in Demo

- **“pv link” is used for all connections because it needs to know the name of the PV that is not connecting in order to white out the corresponding widget.**



# CA Link Commands

- **Syntax**

```
pv link tclVars iocVars [<timeout>]
```

- **Example**

```
pv link current S35DCCT:currentCC 10
set endtime [expr [clock seconds] + 10]
set connected 0
while {[clock seconds] <= $endtime} {
    update
    pv stat current currentStat
    if {$currentStat(state) == “OK”} {
        set connected 1
    }
}
if {$connected} {
    puts $current
}
```

# CA Link Commands

- **Syntax**

```
pv linkw tclVars iocVars [<timeout>] [-blunderAhead]
```

- **Example #1**

```
pv linkw current S35DCCT:currentCC
puts $current
```

- **Example #2**

```
set iocVars “Mt:S:NumBucketsFilledAI NoOfShuttersOpenA”
pv linkw “nbuckets nshutters” $iocVars
puts “nbuckets=$nbuckets nshutters=$nshutters”
```

# CA Link Commands

- **Syntax**

  ```
  pv unlink tclVars
  ```

- **Example**

  ```
  pv linkw current S35DCCT:currentCC
  puts $current
  pv unlink current
  ```

# CA Get Commands

- **When should you use get instead of getw?**
  - When you what to keep the GUI responsive and do not care about the accuracy of the value.
    - *The may be useful when the variable is linked to a widget for display purposes only.*
- **When should you use getw instead of get?**
  - When value of the PV is important.
    - *When using get instead of getw there is no good way of knowing when the PV value has been read.*
  - When ease of use is desirable.
    - *It blocks execution until all the PVs have responded.*

# CA Get Commands

- **Syntax**

  ```
  pv get tclVars [<timeout>]
  ```

- **Example**

  ```
  pv linkw current S35DCCT:currentCC
  …
  pv get current
  …
  puts $current
  ```

# CA Get Commands

- **Syntax**

  ```
  pv getw tclVars [<timeout>]
  ```

- **Example**

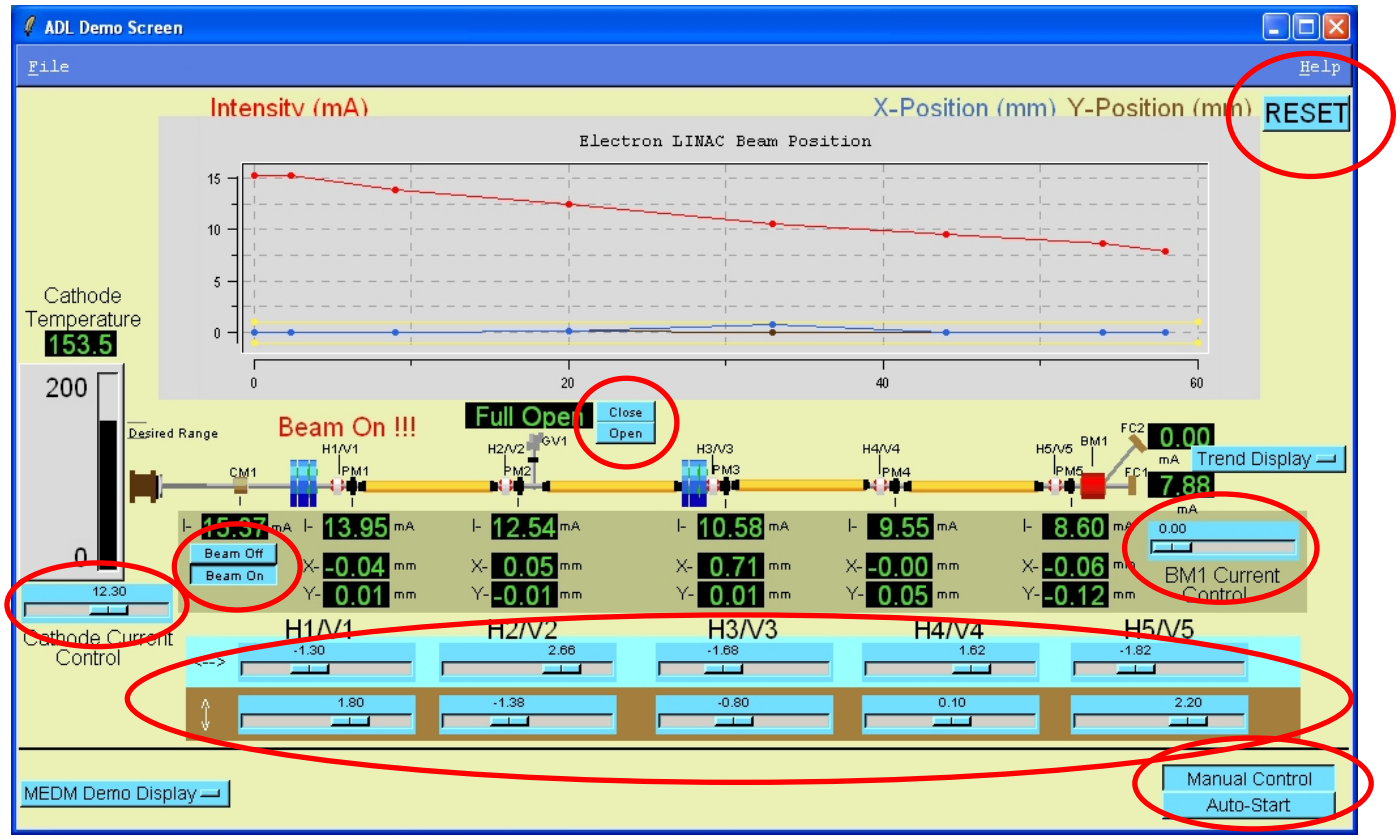
  ```
  pv linkw current S35DCCT:currentCC
  …
  pv getw current
  puts $current
  ```

# CA Put Commands

- **When should you use put?**
  - Require notification from the IOC that record processing has completed. However it will not block execution.
- **When should you use putw?**
  - Require notification from the IOC that record processing has completed. It will block execution until record processing has completed.
- **When should you use putq?**
  - When there is no need to wait for notification from the IOC that record processing has completed.

# CA Put Commands in Demo

- **“pv putq” is used with entry boxes, scales, buttons and radio buttons**



# CA Put Commands

- **Syntax**

  ```
  pv put tclVars [<timeout>]
  ```

- **Example**

  ```
  pv linkw current win32:cathodeCurrentC
  set current 12.0
  pv put current
  ```

# CA Put Commands

- **Syntax**

  ```
  pv putw tclVars [<timeout>]
  ```

- **Example**

  ```
  pv linkw current win32:cathodeCurrentC
  set current 12.0
  pv putw current
  ```

# CA Put Commands

- **Syntax**

  ```
  pv putq tclVars [<timeout>]
  ```

- **Example**

  ```
  pv linkw current win32:cathodeCurrentC
  set current 12.0
  pv putq current
  ```

# CA Monitor Commands

- **When should you use mon instead of umon?**
  - When the Tcl variable is not linked to a Tk widget.
    - *The value of the variable is only updated when it is read.*
- **When should you use umon instead of mon?**
  - When a Tk widget is used to display the value of the variable.
    - *The widget will update at the same time as the variable.*
- **When should you use cmon?**
  - Used to stop monitoring a PV.

# *CA Monitor Commands in Demo*

- **“pv mon” is used with labels, bars, plots, text boxes**
  - Every second a procedure is run to update the widgets
  - This method reduces excess screen updates for readback PVs
- **“pv umon” is used with entry boxes, scales, radio buttons**
  - Widgets are updated instantly because the can be changed by the user.

# CA Monitor Commands

- **Syntax**

  ```
  pv mon tclVars [script]
  ```

- **Example**

  ```
  pv linkw voltage L1:RG2:KIK:PFNVoltageAI
  pv mon voltage {puts “voltage=$voltage”}
  ```

# CA Monitor Commands

- **Syntax**

  ```
  pv umon tclVars [script]
  ```

- **Example**

  ```
  pv linkw voltage L1:RG2:KIK:PFNVoltageAI
  label .voltage –textvariable voltage
  pack .voltage
  pv umon voltage {puts “voltage=$voltage”}
  ```

# CA Monitor Commands

- **Syntax**

  ```
  pv cmon tclVars
  ```

- **Example**

  ```
  pv linkw voltage L1:RG2:KIK:PFNVoltageAI
  pv umon voltage {puts “voltage=$voltage”}
  …
  pv cmon voltage
  ```

# CA Info Command

- **When would you use the info command?**
  - To obtain information in the form of lists of lists about linked Tcl channel access connections.
  - That information could include
    - *State of communication link*
    - *Severity as defined by EPICS*
    - *Status as defined by EPICS*
    - *Units as defined by EPICS*
    - *Choices for enumerated PVs*
    - *IOC name*
    - *Upper and lower limits*
    - *Read and write access privileges*

# CA Info Command

- **Syntax**

  ```
  pv info tclVars <requestedInfo>
    <requestedInfo> - state, severity, status, time,
    units, name, choices, hopr, lopr, drvh, drvl, hihi,
    hi, lolo, lo, precision, ioc, access, size
  ```

- **Example**

  ```
  pv linkw voltage L1:RG2:KIK:PFNVoltageAI
  pv info voltage “lolo hihi”
  ```

# *CA Stat Command*

- **When would you use the stat command?**
  - To obtain state, status, severity and time information about a PV as elements of a Tcl array.
  - Some may find the stat command to be more convenient than the info command

# CA Stat Command

- **Syntax**

  ```
  pv stat tclVar <associative_array_name>
  ```

- **Example**

  ```
  pv linkw voltage L1:RG2:KIK:PFNVoltageAI
  pv stat voltage v
  puts “$v(status) $v(state) $v(severity)”
  ```

# CA toggleErrorMode command

- To keep backward compatibility with previous versions of the CA extension the link, get, put, mon and stat commands by default return 0 on successful completion and 1 when an error occurs. The error message is then stored in a global variable called “errorCode.”
- The toggleErrorMode command toggles between standard Tcl error handling and backward compatibility mode.
- By using standard Tcl error handling it is possible to use the Tcl “catch” command to trap errors.

# *SDDS (Self Describing Data Sets)*

- **SDDS is the file format that we use to store and process data logs of process variables and various other types of data.**
- **Stable, general purpose file protocol**
- **Generic tools that operate on SDDS files**
- **EPICS tools that are configured by SDDS files**
- **Libraries for working with such files**
- **Multi-platform and open-source**
  - Solaris, Linux, Windows, OS X, VxWorks
- **Supported languages**
  - C/C++, Tcl/Tk, Python, Java, IDL, MATLAB, FORTRAN

# *Why Use Self Describing Data?*

- **Programs that use it are much more robust and flexible**
  - Check existence, data-type, units of data instead of crashing or doing something inappropriate
  - Respond appropriately to the data that is provided
    - *Exit and warn user*
    - *Use defaults for missing data*
  - Data does not become obsolete when the program is upgraded
- **Data sets can evolve without breaking applications**
- **Multiple uses for one data set are possible**
  - Helps maintain consistent configuration of multiple applications

# *SDDS File Protocol*

- **Consists of**
  - A header describing what the file contains.
  - One or more pages containing
    - *Parameter values*
    - *Column values*
    - *Array values*
- **The header defines the name, type, and units for the parameters, columns and arrays. Where the type can be a string, character, double, float, long or short.**
- **Includes support for binary, ASCII and compressed storage modes.**

# *SDDS Toolkits*

- **Without the Toolkits, SDDS would just be another boring file format**
- **Toolkit is inspired by UNIX**

- **UNIX**
  - Everything is a file
  - Programs are “filters” operating on ASCII streams
  - Pipes allow sequencing filters arbitrarily

- **SDDS**
  - Everything is a self-describing file
  - Programs are operators that transform datasets
  - Pipes allow sequencing operators arbitrarily

# *SDDS and Tcl/Tk*

- **SDDS and Tcl/Tk complement each other**
- **Tcl/Tk is a good language for GUIs, but**
  - Lacks data management capabilities
  - Not great for computation
- **SDDS offers data management, analysis, and computation, but**
  - Is not a programming language
  - Has commandline user interface

# SDDS Extension

- Loading the SDDS extension into oagwish (basic method)

```
> oagwish
% package require sdds
1.0
```

- Loading the SDDS extension into oagwish (standard method)

```
#!/bin/sh
# \
exec oagwish “$0” “$@”
set auto_path [linsert $auto_path 0 /usr/local/oag/apps/lib/$env(HOST_ARCH)
set auto_path [linsert $auto_path 0 /usr/local/oag/lib_patch/$env(HOST_ARCH)
APSApplication . –name “Demo Script”
```

# *Overview of SDDS Extension Commands*

- **load**
  - Used to load SDDS files into a Tcl array
- **save**
  - For saving SDDS files from a Tcl array
- **Older and superseded commands**
  - open, close
  - getParameter, getColumn, getArray
  - getParameterInformation, getColumnInformation, getArrayInformation
  - getNames
  - sddsDefineParameter, sddsDefineColumn, sddsDefineArray
  - writeLayout, startPage, writePage
  - setParameter, setColumn, setRowValues, setArray

# SDDS Load and Save Commands

- **Syntax**

  ```
  sdds load <fileName> <tclArrayName>
  sdds save <fileName> <tclArrayName>
  ```

- **Contents of Tcl array**
  - (Layout.DataMode.Mode) – “ascii” or “binary”
  - (Layout.LinesPerRow) – integer
  - (Description.Contents) – string
  - (Description.Text) – string
  - (ParameterNames) – list of parameter names
  - (ArrayNames) – list of array names
  - (ColumnNames) – list of column names

# *SDDS Load and Save Commands*

- **Contents of Tcl array (continued)**
  - (ParameterInfo.{name}) – list similar to:
    “name Time symbol {} units {} description {} format_string {} type SDDS_LONG fixed_value {}”
  - (Parameter.{name}) – list where each element represents the parameter value for a different page
  - (ArrayInfo.{name}) – list similar to:
    “name Direction symbol {} units {} description {} format_string {} type SDDS_DOUBLE dimensions 2”
  - (Array.{name}) - List where each element represents the array data for a different page. The data for each page is a list that alternates between an index and the corresponding array value.

# *SDDS Load and Save Commands*

- **Contents of Tcl array (continued)**
  - (ColumnInfo.{name}) – list similar to "name Days symbol {} units {} description {} format_string {} type SDDS_STRING"
  - (Column.{name}) - List where each element represents the column data for a different page. The data for each page is separated into different elements for each row.

# SDDS Load Command

- **Example**

```
sdds load foo.sdds bar
foreach name $bar(ParameterNames) {
    set page 1
    foreach data $bar(Parameter.$name) {
        puts “page $page, $name = $data”
        incr page
    }
}
foreach name $bar(ColumnNames) {
    set page 1
    foreach data $bar(Column.$name) {
        set row 1
        foreach item $data {
            puts “page $page, row $row, $name = $item”
            incr row
        }
        incr page
    }
}
```

# SDDS Save Command

- **Example**

```
set now [clock seconds]
set tomorrow [expr $now + 86400]

set bar(Description.Text) “Example file”
set bar(Description.Contents) “Random data”

set bar(ParameterNames) “Time Date”
set bar(ParameterInfo.Time) “type SDDS_LONG”
set bar(Parameter.Time) [list $now $tomorrow]
set bar(Parameter.Date) [list [clock format $now] [clock format $tomorrow]]

set bar(ColumnNames) “Days Colors”
set bar(Column.Days) [list “Sun Mon Tue” “Wed Thur Fri Sat”]
set bar(Column.Colors) [list “Red Pink Blue” “Brown Black Green White”]

if {[catch {sdds save foo.sdds bar} result]} {
    puts stderr “unable to save foo.sdds: $result”
    exit
}
```

# OAG Tcl/Tk Documentation on the Web

- www.aps.anl.gov/asd/oag/documentation.shtml
  - HTML, postscript and PDF
  - APS Tcl/Tk Library and Interpreter Extensions
  - APS CA Tcl/Tk Extension
  - APS SDDS Tcl/Tk Extension

# *Obtaining and Installing OAG Tcl/Tk*

- **www.aps.anl.gov/asd/oag/oagPackages.shtml**
  - Prebuilt installation packages for Windows and Linux
  - Source code available
    - *Requires EPICS and SDDS packages to be built first.*

# *Conclusions*

- **With 10 years of development and a dedicated OAG staff, OAG Tcl/Tk has become a strong general purpose tool that others can use to avoid excessive development time worrying about the underlying structure.**
- **With the CA and SDDS extensions it is possible to create complex applications well suited for the APS environment.**